



# Adding Parallelism to HPC Applications using Reveal

DOE Centers of Excellence Performance Portability Meeting

Heidi Poxon  
Technical Lead  
Programming Environment  
Cray Inc.

April 2016

# Legal Disclaimer



*Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.*

*Cray Inc. may make changes to specifications and product descriptions at any time, without notice.*

*All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.*

*Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.*

*Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.*

*Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.*

*The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.*

*Copyright 2016 Cray Inc.*



# When to Move to a Hybrid Model

- **When code is network bound**
  - Increased MPI collective and point-to-point wait times
- **When MPI starts leveling off**
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- **When contention of shared resources increases**



# Approach to Adding Parallelism

## 1. Identify key high-level loops

- Determine where to add additional levels of parallelism

## 2. Perform parallel analysis and scoping

- Split loop work among threads

## 3. Add OpenMP layer of parallelism

- Insert OpenMP directives

## 4. Analyze performance for further optimization, specifically vectorization of innermost loops

- We want a performance-portable application at the end

# The Problem – How Do I Parallelize This Loop?

- How do I know this is a good loop to parallelize?
- What prevents me from parallelizing this loop?
- Can I get help building a directive?

```

subroutine sweepz
...
do j = 1, js
  do i = 1, isz
    radius = zxc(i+mypez*isz)
    theta = zyc(j+mypey*js)
    do m = 1, npez
      do k = 1, ks
        n = k + ks*(m-1) + 6
        r(n) = recv3(1,j,k,i,m)
        p(n) = recv3(2,j,k,i,m)
        u(n) = recv3(5,j,k,i,m)
        v(n) = recv3(3,j,k,i,m)
        w(n) = recv3(4,j,k,i,m)
        f(n) = recv3(6,j,k,i,m)
      enddo
    enddo
  ...
  call ppmlr
  do k = 1, kmax
    n = k + 6
    xa(n) = zza(k)
    dx(n) = zdz(k)
    xa0(n) = zza(k)
    dx0(n) = zdz(k)
    e(n) = p(n) / (r(n)*gamm)+0.5 &
      * (u(n)**2+v(n)**2+w(n)**2)
  enddo
  call ppmlr
...
enddo
enddo

```

```

subroutine ppmlr

call boundary
call flatten
call paraset(nmin-4, nmax+5, para, dx, xa)

call parabola(nmin-4,nmax+4,para,p,dp,p6,p1,flat)
call parabola(nmin-4,nmax+4, para,r,dr,r6,r1,flat)
call parabola(nmin-4,nmax+4,para,u,du,u6,ul,flat)

call states(p1,ul,r1,p6,u6,r6,dp,du,dr,plft,ulft,&
  rlft,prgh,urgh,rrgh)
call riemann(nmin-3,nmax+4,gam,prgh,urgh,rrgh,&
  plft,ulft,rlft pmid umid)
call evolve(umid, pmid) ← contains more calls

call remap ← contains more calls

call volume(nmin,nmax,ngeom,radius,xa,dx,dvol)

call remap ← contains more calls

return
end

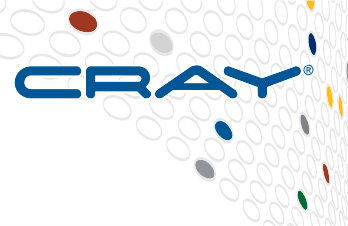
```

# Hybridization Step 1: Loop Work Estimates

*Gather loop statistics using CCE and the Cray performance tools to determine which loops have the most work*

- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop
- **Provides the following statistics**
  - Min, max and average trip counts
  - Inclusive time spent in loops
  - Number of times a loop was executed

# Example Loop Work Estimates



**Table 2:** Loop Stats by Function (from `-hprofile_generate`)

Loop Incl Time Total	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
8.995914	100	25	0	25	sweepy_.LOOP.1.li.33
8.995604	2500	25	0	25	sweepy_.LOOP.2.li.34
8.894750	50	25	0	25	sweepz_.LOOP.05.li.49
8.894637	1250	25	0	25	sweepz_.LOOP.06.li.50
4.420629	50	25	0	25	sweepx2_.LOOP.1.li.29
4.420536	1250	25	0	25	sweepx2_.LOOP.2.li.30
4.387534	50	25	0	25	sweepx1_.LOOP.1.li.29
4.387457	1250	25	0	25	sweepx1_.LOOP.2.li.30
2.523214	187500	107	0	107	riemann_.LOOP.2.li.63
1.541299	20062500	12	0	12	riemann_.LOOP.3.li.64
0.863656	1687500	104	0	108	parabola_.LOOP.6.li.67

# View Source and Optimization Information



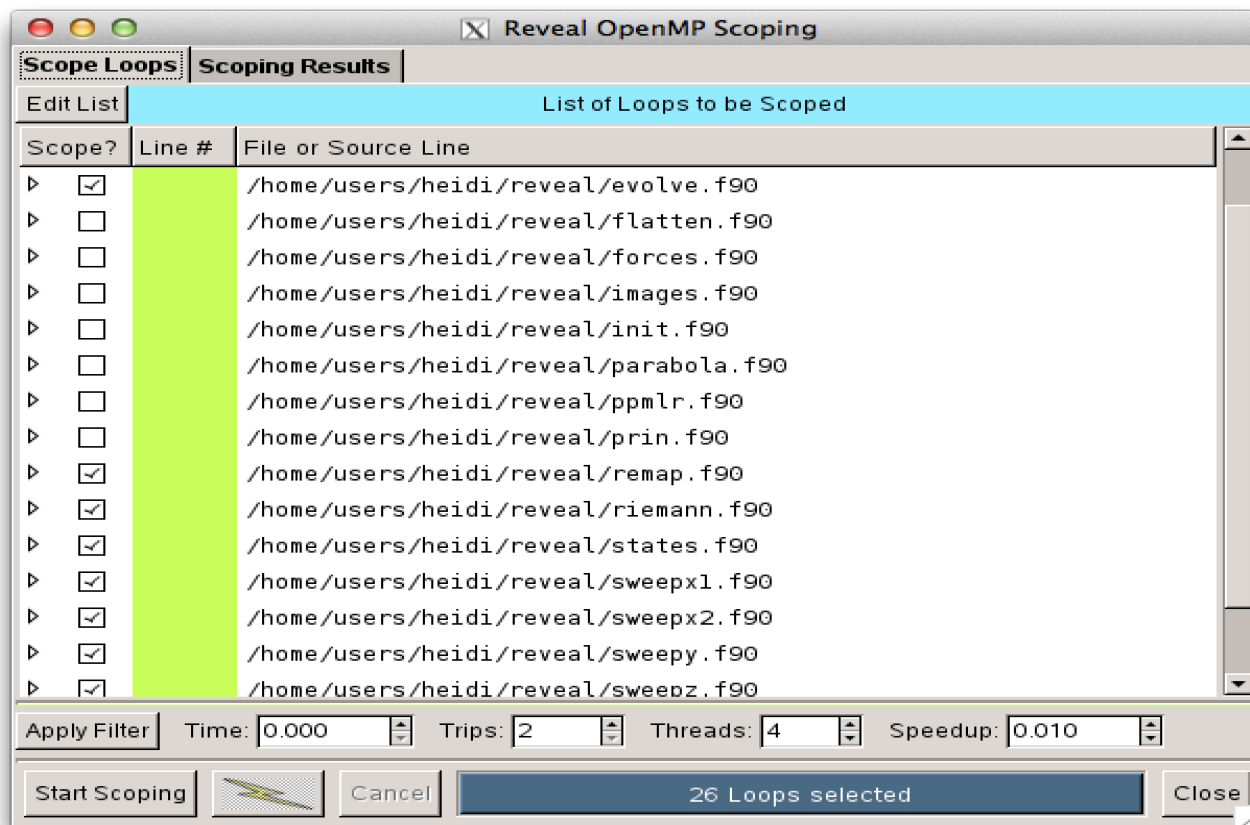
The screenshot displays the Reveal application interface, which is used for viewing source code and optimization information. The main window is titled "Reveal" and shows the source code of a file named "parabola.f90" located at "/home/users/heidi/reveal/parabola.f90". The code is displayed in a monospaced font, with line numbers on the left. The code includes a loop starting at line 67, which is highlighted in green. The loop body contains several assignments and arithmetic operations. The code is as follows:

```
66  
67 do n = nmin, nmax  
68   deltaa(n) = ar(n) - al(n)  
69   a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))  
70   scrch1(n) = (ar(n) - a(n)) * (a(n) - al(n))  
71   scrch2(n) = deltaa(n) * deltaa(n)  
72   scrch3(n) = deltaa(n) * a6(n)  
73 enddo  
74  
75 do n = nmin, nmax  
76   if(scrch1(n) <= 0.0) then  
77     ar(n) = a(n)  
78     al(n) = a(n)  
79   endif
```

The left sidebar shows the "Navigation" pane, which lists the "Loop Performance" for various loops. The "PARABOLA@67" loop is selected, showing its performance metrics and instances. The "Info" pane at the bottom shows the "Info - Line 67" message, which states: "A loop starting at line 67 was fused with the loop starting at line 53." The right sidebar shows the "Loopmark Legend" pane, which lists various optimization marks and their meanings. The marks include: A (Pattern Matched), C (Collapsed), D (Deleted), E (Cloned), G (Accelerated), I (Inlined), II (Not Inlined), L (Loop), M (Multithreaded), R (Region), S (Scoping Analysis), V (Vectorized), a (Atomic Memory Operation), b (Blocked), c (Conditional and/or Computed), f (Fused), g (Partitioned), i (Interchanged), l (Non-blocking Remote Transfer), p (Partial), r (Unrolled), s (Shortloop), and w (Unwound).



# Hybridization Step 2: Scope Selected Loop(s)



# Review Scoping Results



Loops with scoping information are flagged. Red needs user assistance

Navigation: Loop Performance

- 4.0778 SWEEPY@35
- 4.0773 SWEEPY@36
- 4.0529 SWEEPX1@31
- 4.0526 SWEEPX1@32
- 4.0425 SWEEPX2@31
- 4.0423 SWEEPX2@32
- 3.8576 SWEEPZ@51
- 3.8573 SWEEPZ@52
- 2.2068 RIEMANN@63
- 1.2299 RIEMANN@64
- 0.8068 PARABOLA@67
- 0.5429 PARABOLA@44
- 0.5331 PARABOLA@75
- 0.4244 REMAP@83
- 0.3341 PARABOLA@30
- 0.2966 PARABOLA@84
- 0.2915 PARABOLA@53
- 0.2287 RIEMANN@44
- 0.2028 PARABOLA@36
- 0.2009 PARABOLA@117
- 0.1858 PARABOLA@24
- 0.1847 SWEEPY@86
- 0.1771 STATES@64
- 0.1723 EVOLVE@70
- 0.1638 REMAP@111
- 0.1619 PARABOLA@129
- 0.1070 PARABOLA@139
- 0.0938 SWEEPZ@120
- 0.0936 SWEEPZ@121
- 0.0930 SWEEPZ@122
- 0.0925 SWEEPX1@59
- 0.0901 SWEEPZ@22
- 0.0898 SWEEPZ@23
- 0.0892 STATES@50
- 0.0880 SWEEPZ@105

Source: /home/users/heidi/reveal/sweep

```
50 call ppmlr
51 do i = 1, isz
52   radius = zxc(i+mypez*isz)
53   theta = zyc(j+mypey*js)
54   stheta = sin(theta)
55   radius = radius * stheta
56
57
58 ! Put state variables into 1D arrays, padding with 6 ghost zones
59 do m = 1, npez
60   do k = 1, ks
61     n = k + ks*(m-1) + 6
62     r(n) = recv3(1, j, k, i, m)
63     p(n) = recv3(2, j, k, i, m)
64     u(n) = recv3(5, j, k, i, m)
65     v(n) = recv3(3, j, k, i, m)
66     w(n) = recv3(4, i, k, i, m)
```

Info - Line 51

- A loop starting at line 51 was scoped with errors. See Scoping Tool for more information.
- "ppmlr" (called from "sweep2") was not inlined because I/O was detected in "Volume".
- "ppmlr" (called from "sweep2") was not inlined because the enclosing loop body did not completely flatten.
- A loop starting at line 105 is flat (contains no external calls).
- A loop starting at line 105 was not vectorized because it does not map well onto the target architecture.
- A loop starting at line 105 was unrolled 8 times.
- A loop starting at line 51 was not vectorized because it contains a call to subroutine "ppmlr" on line 81.
- A loop starting at line 52 was not vectorized because it contains a call to subroutine "ppmlr" on line 81.
- A loop starting at line 59 is flat (contains no external calls).
- A loop starting at line 59 was not vectorized because a better candidate was found at line 60.
- A loop starting at line 60 is flat (contains no external calls).
- A loop starting at line 60 was not vectorized because it does not map well onto the target architecture.
- A loop starting at line 60 was unrolled 8 times.
- A loop starting at line 71 is flat (contains no external calls).
- A loop starting at line 71 was vectorized.

home/users/heidi/reveal/vhone\_loops.ap2 loaded.

Scope Loops Scoping Results

sweepz.f90: Loop@51

Call or I/O at line 81 of sweepz.f90

- 4. /home/users/heidi/reveal/volume.f90:34
- 3. /home/users/heidi/reveal/evolve.f90:21
- 2. /home/users/heidi/reveal/ppmlr.f90:73
- 1. /home/users/heidi/reveal/sweepz.f90:81

Call or I/O at line 81 of sweepz.f90

4. /home/users/heidi/reveal/volume.f90:34

Name	Type	Scope	Info
wl@remap_1	Scalar	Unresolved	FAIL: Possible recurrence involving this object.
xa	Array	Unresolved	FAIL: Possible resolvable recurrence involving this object.
xa0	Array	Unresolved	FAIL: Possible resolvable recurrence involving this object.
i	Scalar	Private	WARN: LastPrivate of array may be very expensive.
j	Scalar	Private	FAIL: Possible recurrence involving this object.
k	Scalar	Private	FAIL: Possible resolvable recurrence involving this object.
m	Scalar	Private	WARN: LastPrivate of array may be very expensive.
n	Scalar	Private	FAIL: Possible recurrence involving this object.
stheta	Scalar	Private	WARN: LastPrivate of array may be very expensive.
theta	Scalar	Private	FAIL: Possible resolvable recurrence involving this object.
gamm	Scalar	Shared	
isz	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
mypey	Scalar	Shared	

First/Last Private

☐ Enable FirstPrivate

☐ Enable LastPrivate

Find Name:

Insert Directive Show Directive

Close

Parallelization inhibitor messages are provided to assist user with analysis

# Review Scoping Results (2)



Reveal OpenMP Scoping

Scope Loops | Scoping Results

sweepx2.f90: Loop@28

Call or I/O at line 55 of sweepx2.f90  
4: /lus/scratch/heidi/demo/reveal/volume.f90:34  
3: /lus/scratch/heidi/demo/reveal/evolve.f90:21  
2: /lus/scratch/heidi/demo/reveal/ppmlr.f90:49

Name	Type	Scope	Info
ar@parabola_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
da@parabola_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
delta@remap_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
dvol	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
dx	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
dx0	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
e	Array	Unresolved	FAIL: Possible recurrence involving this object.

First/Last Private:  
☐ Enable FirstPrivate  
☐ Enable LastPrivate

Reduction:  
None

Find Name:

'I' identifies variables that reside in functions within the loop

# Review Scoping Results (3)



Reveal OpenMP Scoping

Scope Loops | Scoping Results

sweepy.f90: Loop@35

Call or I/O at line 62 of sweepy.f90  
4: /home/users/heidi/reveal/volume.f90:34  
3: /home/users/heidi/reveal/evolve.f90:21

Name	Type	Scope	Info
ks	Scalar	Shared	
mypey	Scalar	Shared	
ndim	Scalar	Shared	
npey	Scalar	Shared	
recv1	Array	Shared	
send2	Array	Shared	
svel <b>RI</b>	Scalar	Shared	<b>WARN:</b> atomic reduction operator required unless reduction fully
zdy	Array	Shared	
zxc	Array	Shared	
zya	Array	Shared	

First/Last Private  
☐ Enable FirstPrivate  
☐ Enable LastPrivate

Reduction:  
None

Find Name:

Insert Directive Show Directive Close

Reveal identifies calls that prevent parallelization

Reveal identifies shared reductions down the call chain

# Hybridization Step 3: Generate OpenMP Directives

! Directive inserted by Cray Reveal. May be incomplete.

```
!$OMP parallel do default(none) &
!$OMP& unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w, &
!$OMP& xa,xa0) &
!$OMP& private (i,j,k,m,n,$_n,delp2,delp1,shock,temp2,old_flat, &
!$OMP& onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn, &
!$OMP& ekin) &
!$OMP& shared (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, &
!$OMP& recv1,send2,zdy,zxc,zya)
do k = 1, ks
do i = 1, isy
radius = zxc(i+mypey*isy)
```

! Put state variables into 1D arrays, padding with 6 ghost zones

```
do m = 1, npey
do j = 1, js
n = j + js*(m-1) + 6
r(n) = recv1(1,k,j,i,m)
p(n) = recv1(2,k,j,i,m)
u(n) = recv1(4,k,j,i,m)
v(n) = recv1(5,k,j,i,m)
w(n) = recv1(3,k,j,i,m)
f(n) = recv1(6,k,j,i,m)
enddo
enddo
```

```
do j = 1, jmax
n = j + 6
```

Reveal generates  
OpenMP directive with  
illegal clause marking  
variables that need  
addressing

# Hybridization Step 4: Performance Analysis



Choose "Compiler Messages" view to access message filtering, then select desired type of message

The screenshot shows the vhone.pl application interface. On the left, the "Navigation" pane is open to the "Compiler Messages" view, showing a tree of messages. The "Source" pane on the right displays the source code for the file `/home/users/heidi/reveal/riemann.f90`. The code is a Fortran program with a loop starting at line 64. The "Info" pane at the bottom provides details about the loop starting at line 64, indicating it is flat and contains no external calls, and that a loop starting at line 64 was not vectorized because a recurrence was found on "pmid" at line 77.

**Navigation**

- Compiler Messages
- Not Vectorized
- All
- line 123
- images.f90
- line 149
- init.f90
  - line 113 (0.000 sec)
  - line 114 (0.000 sec)
  - line 153 (0.000 sec)
  - line 154 (0.000 sec)
  - line 139
- prin.f90
  - line 125 (0.006 sec)
  - line 42 (0.000 sec)
  - line 43 (0.000 sec)
  - line 127 (0.000 sec)
  - line 128 (0.000 sec)
  - line 129 (0.000 sec)
  - line 104 (0.000 sec)
- riemann.f90
  - line 63 (0.387 sec)
  - line 64 (0.224 sec)
- sweepx1.f90
  - line 31 (4.053 sec)
  - line 32 (4.053 sec)
  - line 59 (0.093 sec)

**Source** - /home/users/heidi/reveal/riemann.f90

```
62
63 do l = lmin, lmax
64 do n = 1, 12
65   pmold(l) = pmid(l)
66   wlft(l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67   wrgh(l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68   wlft(l) = clft(l) * sqrt(wlft(l))
69   wrgh(l) = crgh(l) * sqrt(wrgh(l))
70   zlft(l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71   zrgh(l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72   zlft(l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)
73   zrgh(l) = zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)
74   umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75   umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76   pmid(l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (
77   pmid(l) = max(smallp,pmid(l))
78   if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79 enddo
```

**Info** - Line 64

- A loop starting at line 64 is flat (contains no external calls).
- A loop starting at line 64 was not vectorized because a recurrence was found on "pmid" at line 77.

vhone.pl loaded. vhone\_loops.ap2 loaded.

# Summary



- **Reveal can be used to simplify the task of adding OpenMP to MPI programs**
- **The result is performance portable code**
  - Programs can be built with any compiler that supports OpenMP
- **Can be used as a stepping stone for codes targeted for nodes with higher core counts and as the first step in adding directives to applications to target GPUs**